허프만 코드를 이용한 압축 프로그램 실험

2016706018 박건희

본 실험은 허프만 코드를 이용한 인코딩 파일의 압축률이 어느정도 되는지 진행하였다.

실험에서 압축률은 다음과 같이 원본파일크기 대비 압축된 비율을 [%]단위로 정의하였다.

압축률 = (1-압축파일크기/원본크기)\*100 [%]

**실험에서는 과제사항에 명시된 알파벳, 숫자와 특정 문자들 이외에도 범용성을 고려해 텍스트파일에 사용 가능한 아스키코드 전체가 인코딩 가능하도록 구현했다.**

**인코딩 체계**

인코딩 체계 헤더는 hfb파일을 압축해제시 헤더를 먼저 읽고, 그 뒤 변환된 바이너리들을 읽는 점에서 몇 가지 부분을 고려하였다.

1. 인코딩 체계의 헤더의 끝과 허프만 코드로 변환된 바이너리의 시작의 구분을 어떻게 할지

2. 허프만 코드로 변환된 바이너리가 어디까지 유효 한지

3. 디코더에서 인코딩된 바이너리를 텍스트로 변환할 때 어떤 정보들이 필요한지

위 3가지가 충족되어야 변환된 바이너리를 문제없이 텍스트로 복원 가능하다 판단했다.

파일 입출력시 쓰이는 최소 단위는 바이트(=8bit) 이기 때문에, 헤더파일의 바이트 순서로 정보를 구분 지었다.

가장 첫번째 바이트는 hfb 파일 마지막 바이트의 유효 bit이다.

마지막 바이트의 유효 bit란 인코딩 체계의 다음부터는 텍스트가 허프만 코드 바이너리로 변환되어 입력되는데, 이 경우 예를 들면, a=11 b=101의 허프만 코드가 부여된 텍스트 abba의 경우는 1110110111의 바이너리가 되는데 앞서 언급한대로, 파일 입출력의 최소 단위는 바이트이기 때문에 8bit씩 나눠서 저장을 하게 된다.

이 경우 총 10개의 바이너리이고 8bit씩 나누면, 11101101과 11000000으로 나뉘게 되어 총 2바이트의 저장공간을 할당하게 된다. 이때 두번째 바이트의 MSB로부터 2bit까지만 텍스트 abba의 정보이고 나머지 6bit인 0으로 채워진 공간은 디코딩시 문자로 변환을 할 필요가 없는 부분이다.

하지만 만약, 이 부분을 언급을 안 하면 마지막에 6bit를 읽고, 허프만코드가 0으로만 이루어진 문자로 변환을 할 가능성이 있기 때문에 파일의 끝을 지정하기 위해 마지막 바이트의 유효bit를 언급하여야 한다.

두번째 바이트에는 hfb파일에 사용된 문자 종류의 개수가 입력된다.

만약, ababc와 같이 문자 a, b, c로만 이루어진 텍스트는 총 3개의 문자정보가 필요하기 때문에, 두번째 바이트로 3인 (00000011)이 입력된다.

아스키코드의 범위가 1바이트인 255개보다 작기 때문에 1바이트로 충분히 표현이 가능하다.

두번째 바이트에 문자 종류의 개수를 입력 받는 이유는 세번째 바이트부터는 각 문자에 대한 정보가 차례로 나오는데 몇 번을 반복해야 문자에 대한 정보의 바이트만 전부 입력 받고, 인코딩 체계의 끝이 정의가 되는지 알 수 있기 때문이다.

세번째 바이트부터는 각 문자에 대한 세가지 정보를 담은 바이트가 입력된다.

이 과정을 두번째 바이트로부터 받은 문자 종류의 개수만큼 반복한다.

세가지 정보는 첫째로, 문자가 무엇인지 알 필요가 있기 때문에 문자의 아스키코드 값, 두번째로는 해당 문자의 허프만 코드의 길이, 세번째로는 해당 문자의 허프만 코드이다.

두번째에서 허프만 코드의 길이를 입력 받은 이유는 허프만 코드가 몇 바이트의 공간에 몇 비트만큼 유효하게 존재하는지 알기 위해서다.

예를 들어 설명하면, a라는 문자의 허프만 코드 길이가 10인 1111011111이면 허프만 코드가 8bit의 공간에 전부 들어갈 수 없기에 2바이트의 공간의 사용이 필요하고, 8bit에 들어간 공간 이외에 나머지 2bit가 1바이트에 들어갔을 때 1바이트의 공간에 2bit가 들어있다는 것을 알아야 한다.

즉, 1바이트가 넘는 경우와 바이트내 몇 bit만큼이 코드로 할당 된지 알기 위해서는 필수적으로 알아야 하는 정보이다.

세번째로는 문자의 허프만 코드이다.

케이스를 길이가 8을 넘는 경우와 8을 넘지 않는 경우로 나눌 수 있는데

만약 11011의 허프만 코드이면 00011011로서 1바이트 내로 저장이 되지만, 1111011111과 같이 8bit를 넘는 10bit인 경우 11110111과 00000011로 나누어 총 2바이트의 공간할당이 필요하다.

예시로, 아래와 같은 허프만 코드를 가진 testcase0.txt에서



바이너리 뷰어를 통해 testcase0.hfb의 바이너리를 분석시



첫번째 바이트에는 마지막 바이트의 유효 bit인 1,

두번째 바이트에는 문자의 종류가 a와 z뿐이기 때문에 2

세번째 바이트부터는 순차적으로

a의 아스키 코드인 97, a의 허프만 코드의 길이 1, a의 허프만 코드 0

z의 아스키 코드인 122, z의 허프만 코드의 길이 1, z의 허프만 코드 1

이렇게 총 8바이트로 인코딩 헤더가 구성되었고 이후로는 텍스트가 허프만코드를 기반으로 변환된 바이너리가 입력되어 있다.

헤더 이후 인코딩은 1바이트의 공간에 MSB부터 문자의 허프만 코드를 가능한 만큼 전부 순차적으로 채우고 1바이트를 전부 할당 시 파일에 쓰고, 다음 바이트를 할당 받아서 진행하는 방식으로 하였다.

또한 마지막 바이트에서 파일의 끝을 알려주는 유효bit에 관한 정보를 파일의 첫 바이트에 입력하였다.

**압축률 실험**

압축률은 테스트케이스 5가지를 진행하였다.

테스트케이스는

1. 문자의 종류가 2가지 이하인 경우

2. 실제로 사용되는 텍스트파일

3. 문자의 종류가 많고, 전체 텍스트의 길이가 짧은 경우

4. 텍스트파일의 크기가 크며, 특정 문자의 종류가 파일의 대부분을 차지하는 경우

5. 텍스트파일의 크기가 크며, 모든 문자의 빈도수가 거의 균일한 경우

위 5가지 종류로 구성하였다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | testcase0.txt | testcase1.txt | testcase2.txt | testcase3.txt | testcase4.txt |
| 원본 파일 크기 (Bytes) | 1427 | 4525 | 511 | 150000 | 1000000 |
| 압축 파일 크기  (헤더 미포함)  (Bytes) | 179 | 2726 | 337 | 97085 | 745903 |
| 압축률  (헤더 미포함)  (%) | 87.46 | 39.76 | 34.05 | 37.28 | 25.41 |
| 압축 파일 크기  (헤더포함)  (Bytes) | 187 | 3005 | 549 | 97387 | 746091 |
| 압축률  (헤더포함)  (%) | 86.90 | 33.59 | -7.44 | 35.08 | 25.39 |
| 헤더 크기  (Bytes) | 8 | 279 | 212 | 302 | 188 |

실험 결과를 분석해보면, testcase0의 경우는 문자가 2종류밖에 없기 때문에 허프만 코드가 0과 1의 1bit로 구성되었다.

1바이트(=8bit)의 문자가 1bit로 표현이 가능하게 되어서 약 원본파일 대비 압축파일 용량이 약 8분의1로 줄어들었고, 가장 압축이 잘 되는 케이스이다.

Testcase1의 경우는 소스코드를 텍스트파일로 변환하여 실험한 케이스로, 실제로 사용하는 파일의 경우 어느 정도로 압축이 될지를 확인하기 위하여 진행하였다.

문자의 종류가 많아지면서, 헤더파일의 크기의 비중이 어느정도 높아졌다.

Testcase2는 특수한 케이스인데, 인코딩 체계를 포함하여 압축할 때 헤더 제외 용량은 34%가량 감소했지만, 실제로는 헤더에 의해 용량이 원본보다 7.44% 더 커졌다.

이는 문자의 길이보다 인코딩 헤더가 용량에 더 지배적인 것을 보여주는 지표이다.

인코딩 헤더의 크기는 사용된 문자의 종류의 개수에 지배적이기 때문에 문자의 종류가 많고 텍스트길이가 짧을수록, 헤더를 포함하여 압축할 때 헤더의 용량이 지배적이게 된다.

Testcase3은 150000개의 텍스트이고 특정 문자들의 개수가 다른 문자들의 개수보다 더 많도록 제작된 케이스이다. Testcase0과 마찬가지로, 특정 문자들의 비중이 높기 때문에 압축이 매우 잘 된 케이스이다. 또한 텍스트의 길이가 매우 길어지면서 헤더의 크기 비중이 매우 감소한 것으로 확인된다.

Testcase4는 1000000개의 텍스트이고 랜덤함수를 통해 모든 문자들의 비중이 uniform하게 생성되었다. 비중이 uniform하여 모든 문자들의 허프만 코드 길이가 5인 2가지 경우 제외 나머지가 6이었고, 대략적으로 계산 시 1바이트(=8bit)짜리 문자가 6bit로 표현이 되어 약 25%가량 용량이 줄어들 것으로 예상되었고 실제 지표에서 25%의 용량이 줄어든 것을 확인하였다.

이때 텍스트길이가 헤더의 크기에 비해 매우 지배적이기 때문에, 헤더의 비중이 0.02%밖에 안되는 것을 지표를 통하여 확인 가능하다.

**결론**

허프만 코드를 이용한 압축은 파일의 전체 크기, 파일을 구성하는 문자의 종류 개수, 파일을 구성하는 각 문자의 개수 위 세가지 사항에 의존한다.

1. 파일의 크기가 클수록 인코딩 헤더에 의한 용량이 무시된다.

2. 파일을 구성하는 문자의 종류 개수가 인코딩 헤더의 용량이 높아진다.

3. 파일을 구성하는 각 문자의 종류 개수와 각 문자의 개수에 따라 허프만 코드의 길이가 달라진다. 비중이 비슷할수록 각 문자의 허프만 코드의 길이가 서로 같아진다.